Announcements (February 8)

- Homework #1 due today
- No class this Thursday (February 10)
- Reading assignments this week
  - Generalized search trees (due next Tuesday)
  - "The" Google paper (due next Thursday)

R-trees

- B-tree: balanced hierarchy of 1-d ranges
- R-tree: balanced hierarchy of n-d ranges
R-tree lookup

- Which ranges contain me?

- Problem: search may go down many paths
  - Because regions may overlap
  - No performance guarantee like B-tree

R-tree insertion

Insert $R_9$ into R-tree

- Start from the root
- Pick a region containing $R_9$ and follow the child pointer
  - If none contains $R_9$, pick one and grow it to contain $R_9$
  - Pick the one that requires the least enlargement (why?)

R-tree insertion: split

- If a node is too full, split
- Try to minimize the total area of bounding boxes
  - Exhaustive: try all possible splits
  - Quadratic: "seed" with the most wasteful pair; iteratively assign regions with strongest "preference"
  - Linear: "seed" with distant regions; iteratively assign others as Quadratic
R-tree insertion: split (cont’d)

- Split could propagate all the way up to the root (not shown in this example)

---

R*-tree

- R-tree
  - Always tries to minimize the area of bounding boxes
  - Quadratic splitting algorithm encourages small seeds and possibly long and narrow bounding boxes

- R*-tree (Beckmann et al., SIGMOD 1990)
  - Consider other criteria, e.g.:
    - Minimize overlap between bounding boxes
    - Minimize the margin (perimeter length) of a bounding box
  - Forced reinserts
    - When a node overflows, reinsert “outer” entries
    - They may be picked up by other nodes, thus saving a split

---

R+-tree

- Problem with R-tree

- R+-tree (Sellis et al., VLDB 1987)
  - Regions in non-leaf nodes do not overlap
  - Search only goes down one path
  - Duplicate items in leaves
Review

- Tree-structured indexes
  - ISAM
  - B-tree and variants
  - R-tree and variants
  - Can we generalize? GiST!

Indexing user-defined data types

- Specialized indexes (ABCDEFG trees…)
  - Redundant code: most trees are very similar
  - Concurrency control and recovery especially tricky to get right
- Extensible B-trees and R-trees
  - Examples: B-trees in Berkeley DB, B- and R-trees in Informix
  - User-defined `compare()` function
- GiST (Generalized Search Trees)
  - General (covers B-trees, R-trees, etc.)
  - Easy to extend
  - Built-in concurrency control and recovery

Structure of GiST

Balanced tree of \( \langle p, ptr \rangle \) pairs

- \( p \) is a key predicate that holds for all objects found below \( ptr \)
- Every node has between \( kM \) and \( M \) index entries…
  - \( k \) must be no more than \( \frac{1}{2} \) (why?)
- Except root, which only needs at least two children
- All leaves are on the same level

- User only needs to define what key predicates are
Defining key predicates

- boolean `Consistent(entry entry, predicate query)`
  - Return true if an object satisfying `query` might be found under `entry`
- predicate `Union(set<entry> entries)`
  - Return a predicate that holds for all objects found under `entries`
- real `Penalty(entry entry1, entry entry2)`
  - Return a penalty for inserting `entry2` into the subtree rooted at `entry1`
- (set<entry>, set<entry>) `PickSplit(set<entry> entries)`
  - Given $M+1$ entries, split it into two sets, each of size at least $kM$

Index operations

- Search
  - Just follow pointer whenever `Consistent()` is true
- Insert
  - Descent tree along least increase in `Penalty()`
  - If there is room in leaf, insert there; otherwise split according to `PickSplit()`
  - Propagate changes up using `Union()`
- Delete
  - Search for entry and delete it
  - Propagate changes up using `Union()`
  - On underflow
    - If keys are ordered, can borrow/coalesce in B-tree style
    - Otherwise, reinsert stuff in the node and delete the node

GiST over $R$ ($B^+$-tree)

- Logically, keys represent ranges $[x, y)$
- Query: find keys that overlap with $[a, b)$
- `Consistent(entry, [a, b])`: say entry has key $[x, y)$
  - $x < b$ and $y > a$, i.e., overlap
- `Union(entries): say entries = \{[x_1, y_1]\}
  - $\{\min(x_1), \max(y_1)\}$
- `Penalty(entry1, entry2)`: say they have keys $[x_1, y_1)$ and $[x_2, y_2)$
  - $\max(y_2 - y_1, 0) + \max(x_1 - x_2, 0)$, except boundary cases
- `PickSplit(entries)`
  - Sort entries and split evenly
- Plus a special `Compare(entry, entry)` for ordered keys
Key compression

- Without compression, GiST would need to store a range instead of a single key value in order to support B+-tree
- Two extra methods: Compress/Decompress
- For B+-tree
  - \( \text{Compress}(\text{entry}) \): say entry has key \([x, y)\)
  - \(x\), assuming next entry starts with \(y\), except boundary cases
  - \( \text{Decompress}(x, \text{ptr}) \)
  - \([x, y)\), assuming next entry starts with \(y\), except boundary cases

This compression is lossless: \( \text{Decompress}(\text{Compress}(e)) = e \)

GiST over \(R^2\) (R-tree)

- Logically, keys represent bounding boxes
- Query: find stuff that overlaps with a given box
  - Abusing notation a bit below…
  - \( \text{Consistent}(\text{key\_box}, \text{query\_box}) \)
    - \(\text{key\_box}\) overlaps with \(\text{query\_box}\)
  - \( \text{Union}(\text{boxes}) \)
    - Minimum bounding box of \(\text{boxes}\)
  - \( \text{Penalty}(\text{box}_1, \text{box}_2) \)
    - Area of \( \text{Union}(\text{box}_1, \text{box}_2) \) – area of \(\text{box}_1\)
  - \( \text{PickSplit}(\text{boxes}) \)
    - R-tree algorithms (e.g., minimize total area of bounding boxes)
  - \( \text{Compare}(\text{box}_1, \text{box}_2) \)

GiST over \(P(Z)\) (RD-tree)

- Logically, keys represent sets
- Queries: find all sets that intersect with a given set
  - \( \text{Consistent}(\text{key\_set}, \text{query\_set}) \)
  - \( \text{Union}(\text{sets}) \)
  - \( \text{Penalty}(\text{set}_1, \text{set}_2) \)
  - \( \text{PickSplit}(\text{sets}) \)
  - \( \text{Compare}(\text{set}_1, \text{set}_2) \)
  - \( \text{Compress}/\text{Decompress}: \text{bloomfilters, rangesets, etc.} \)
Next

- Hash-based indexing
- Text indexing